doi: 10.14355/spr.2015.04.008

Analysis of AVF Evaluation Methods for Microprocessor Reliability

Baolong Guo[1&], Guochang Zhou [2], Yufang Cao[3#], Xiang Gao[4], Yunyi Yan[5*]

[1,3,5]School of Aerospace Science and Technology, Xidian University, Xi'an 710071, China

[2,4]China Academy of Space Technology (Xi'an), Xi'an 710000, China

[&] blguo@xidian.edu.cn; [#]yufangxd@163.com; [*]yyyan@xidian.edu.cn

Abstract

Soft error arises from the strike of high-energy particle, and has become a key challenge in microprocessor design. Designers clearly require accurate estimates the error rates to make appropriate cost. A key aspect of the estimation is the analysis of architectural vulnerability factor (AVF). The AVF of a processor structure is defined as the probability that a fault in a processor structure will result in a visible error in the final output of a program, and AVF is one of the most commonly used estimation metrics of a structure's vulnerability. In this paper, we describe the AVF's concept, computing methods and comparison of three models, and we also discuss the directions for future research.

Keywords

Architectural Vulnerability Factor; Microprocessor Reliability; Architecturally Correct Execution

Introduction

Soft errors due to radiation-caused by neutrons in cosmic rays or alpha particles in packaging material are becoming an increasing burden for microprocessor designers. The microprocessor is the core of the electronics components, and once the soft error occurs, it may cause serious damage, and the approaches for protection introduce a significant penalty in performance, power, die size, and design time. Consequently, designers must carefully evaluate the soft error rate of a microprocessor to decide on the appropriate amount of protection necessary for a target market.

If a particle strike causes a bit to flip or a piece of logic to generate a wrong result, we call the bit flip or the wrong result a raw soft error. Fortunately, not all raw soft errors cause the program to fail. For example, a soft error in a functional unit that is not currently processing an instruction or in an SRAM cell that is not storing useful data will not harm the execution. Such an error is said to be masked. Research has shown that there is a large masking effect at the architecture and micro-architecture levels [1, 2, 3, and 6]; e.g., Wang et al report more than 85% masking [3]. Usually, the masking will not cause the program to fail. Hence, an important aspect of fault modeling is how to measure the impact of the masking effect of the fault.

Three common terms are often used to discuss system reliability: failure rate, mean time between failure (MTBF), and failures in time (FIT) [4]. MIBF and FIT are two commonly used units for error rates currently. For example, for its Power4 processor-based systems, IBM targets 1000 years system MTBF for silent data corruption (SDC) errors, 25 years system MTBF for detected recoverable errors (DUE) that result in a system crash, and 10 years system MTBF for DUE errors that result in an application crash [4]. FIT is inversely related to MTBF. One FIT specifies one failure in a billion hours. Thus, 1000 years MTBF equals 114 FIT. The effective FIT rate per bit is influenced by several vulnerability factors. In general, a vulnerability factor indicates the probability that an internal fault in a device's operation will result in an externally visible error [6]. According to the visible system error Mukherjee et al introduced the architectural vulnerability factor (AVF) and used AVF to estimate system reliability.

Computing AVF for all processor structures is a key aspect of such soft error analysis. This paper generalizes the current methods for evaluating the AVF. Firstly, it introduces the AVF's basic concept, then analyzes the evaluation method, discusses their limitations and advances, and suggests the directions for future research.

Architectural Vulnerability Factor and ACE Analysis

Architectural Vulnerability Factor

Mukherjee et al found that not all faults in a micro-architectural structure affect the final outcome of a program. As a result, an estimate based only on raw device fault rates will be pessimistic, leading architects to over-design their processor's fault-handling features. For example, a single bit fault in a branch predictor will not affect the sequence or results of any committed instructions. They call the probability that a fault in a processor structure will result in a visible error in the final output of a program that structure's Architectural vulnerability factor (AVF). For example, a single bit fault in a branch predictor will not affect the executing of any program, thus, the branch predictor's AVF is 0%. In contrast, a single bit fault in the committed program counter will cause the wrong instructions to be executed, almost certainly affecting the program's result. Hence, the AVF for the committed program counter is effectively 100%. The overall error rate of a micro-architectural structure is the product of its raw fault rate and it's AVF. By summing the contributions of all on-chip structures, a processor architect can map the raw fault rate (dictated by process and circuit issues) to an overall processor error rate, and thus determine whether the design meets its error rate goals (set according to the target market).

ACE Analysis

The AVF of a processor structure is defined as the probability that a fault in that structure will result in a visible error in the final output of a program [6]. A bit in which a fault will result in incorrect execution is said to be necessary for architecturally correct execution (ACE); these bits are termed ACE bits. All other bits are Un-ACE bits. An individual bit may be ACE for a fraction of the overall execution cycles and Un-ACE for the rest. Therefore, the AVF of a single bit can be defined as the fraction of cycles that the bit is ACE. For hardware structure H with size B_H (in bits), its AVF over a period of N cycles can be expressed as follows [6]:

$$AVF_H = \frac{\sum_N ACE\ bits\ in\ H}{B_H \times N}$$

For example, if a storage cell contains ACE bits for a million cycles out of an execution of ten million cycles, then the AVF for that cell is 10%.The average AVF of an entire processor can be computed as the weighted average of the AVFs of each structure for systems of reasonable size [5].

Given this equation of AVF, we would like to determine which bits are ACE and which are Un-ACE. However, ACE bits are uneasy to determine. So we desire a conservative (upper-bound) AVF estimate, we firstly assume that all bits are ACE bits unless we can show otherwise. We then identify as many sources of Un-ACE bits as we can. Mukherjee et al has divided the sources of Un-ACE bits into two general categories: micro-architectural un-bits and architectural Un-ACE bits. The processor state bits that cannot influence the committed instruction path are called micro-architectural Un-ACE bits. They can arise from the following four situations: Idle or Invalid State, Mis-speculated State, Predictor Structures, and Ex-ACE State. Architectural Un-ACE bits are those that affect correct-path instruction execution, but only in ways that do not change the output of the system. The five sources of architectural Un-ACE bits are: NOP instructions, Performance-enhancing instructions, Predicated-false instructions, dynamically dead instructions, and Logical masking.

Mukherjee et al introduced lifetime analysis to compute the AVF of a processor's instruction queue and execution units [6]. Lifetime analysis involves dividing up a bit's lifetime during a program execution into ACE and Un-ACE components. The AVF is the fraction of the bit's lifetime during which the bit contained ACE state. To compute the AVF we use Mukherjee et al's conservative assumption that the entire lifetime is ACE and then systematically prove which portion of the bit's lifetime is Un-ACE. Table 1 shows a detailed classification of lifetimes into ACE and Un-ACE components [7]. By definition idle, read-to-write, and write-to-write are Un-ACE. Whether fill-to-read and write-to-read are Un-ACE depends on the read itself. For example, if the read is dynamically dead (its value will never be used in future and, therefore, will not affect the final outcome of a program), then the write-to-read time is Un-ACE. Other examples of Un-ACE reads include those on the wrong-path or those falsely predicated.

Computing AVF

According to the study, the following models can be used to calculate the AVF: (1) Statistical fault injection, (2) Analysis model, (3) Performance model. Section 3.1 describes the study about statistical fault injection. Section 3.2 describes how to compute the AVF using Little's Law. Section 3.3 describes how to compute the AVF using a performance model. Section 3.4 shows the comparison amongst the three models.

TABLE 1 CLASSIFICATION OF LIFETIMES INTO NON-OVERLAPPING ACE OR UN-ACE COMPONENTS. DYNAMICALLY DEAD READS OR WRITES (NOT SHOWN EXPLICITLY) CONVERT ACE INTO UN-ACE COMPONENTS

Processor Structure	**Lifetime Classification**		
	ACE	***Un-ACE***	***Unknown***
Write-through data cache	fill-to-read, read-to-read, write-to-read	idle, fill-to-write, fill-to-evict, read-to-write, read-to-evict, write-to-write, write-to-evict, evict-to-fill	fill-to-end, read-to end, write-to-end
Write-back data cache	fill-to-read,read-to-read,write-to-read,write-to-evict,write-to-end, some of Un-ACE components can be conditionally ACE(see prose)	idle,fill-to-write,fill-to-evict,read-to-write,read-to-evict,write-to-write, evict-to-fill	fill-to-end, read-to end
Data Translation Buffer	fill-to-read, read-to-read	idle, read-to-evict, evict-to-fill	fill-to-end, read-to end
Store Buffer	fill-to-read,fill-to-evict,fill-to-end,read-to-read,read-to-evict,read-to-end	idle, evict-to-fill	none

Statistical Fault Injection

Many studies have used fault injection into hardware RTL models [3, 8, and 10] and demonstrated AVFs of 1%-10% for latches [8]. For example, Wang injected faults into an RTL model of the Alpha 21264 processor and reported AVFs of less than 10% for latches [8]. Wu et al developed a fault injection tool based on VHDL that implemented by simulator technique [9]. The RTL model has all the hardware structures necessary to create a processor, and it's the biggest advantage of using an RTL model. However, statistical fault injection requires simulating a large number of fault cases to provide adequate statistical significance and an RTL model. The RTL model is generally not available during the architectural exploration phase of a microprocessor design project.

Analysis Model

Little's Law [11] as the analysis model can be translated into the equation $N = B \times L$, where N = average number of bits in a box, B = average bandwidth per cycle into the box, and L = average latency of an individual bit through the box. Applying this to ACE bits, we get the average number of ACE bits in a box as the product of the average bandwidth of ACE bits into the box (B_{ace}) times the average residence cycles of an ACE bit in the box (L_{ace}).Thus, we can express the AVF of a structure as:

$$AVF_H = \frac{B_{ace} \times L_{ace}}{total\ number\ of\ bits\ in\ H}$$

In many cases, it is possible to compute the bandwidth of ACE bits into a structure and the average residence cycles of ACE instructions using hardware performance counters, allowing AVF estimation without a simulation model. Using Little's Law, we can compute the average number of ACE bits resident in a structure [11, 12], therefore, the AVF of the structure are usually used in early design without the RTL and performance model.

Performance Model

The main idea of performance model is to determine the object flows through machines which are the ACE bits or Un-ACE bits. According to the lifetime analysis we know the challenge of performance model is how to determine the Un-ACE parts. To compute the AVF of a structure using the equation in Section 2.2, we need the following information:

- sum of all residence cycles of all ACE bits of the objects resident in the structure during the execution of

the program,

- total execution cycles for which we observe the ACE bits' residence time,
- Total number of bits in a hardware structure.

Using a performance model, we can compute all of the information above. We assume objects that carry instruction information along the pipeline. The AVF algorithm can be divided up into three parts. First step: as an instruction flows through different structures in the pipeline, we record the residence time of the instruction in the structure. Then, before the instruction disappears from the machine-either via a commit or via a squash-we update the structures it flowed through with a variety of information, such as the residence cycles, whether the instruction committed, etc. Second step: if the instruction commits, we put the instruction in a post-commit analysis window to determine if the instruction is dynamically dead or if there are any bits that are logically masked. Third step, at the end of the simulation, using the information captured in step 1 and 2, we can easily compute the AVF of a structure.

Currently researchers have extensively studied AVF evaluation methods in the architecture level and put forward several assessment tools [2, 13, 16, and 17]. Xin Fu et al proposed the Sim-SODA (Software Dependability Analysis), a unified framework for estimating microprocessor reliability in the presence of soft errors at the architectural level [13]. They used COOLDOWN and hamming distance one mechanisms introduced [14] to accurately compute AVF. The Sim-SODA framework includes AVF models for instruction queue, register file, cache, TLB, and load/store queue. Yu Cheng et al proposes a Hybrid AVF Evaluation Strategy (HAES) which combines memory access analysis and instruction identification for AVF evaluation [18]. Then the HAES is integrated into a general simulator and an improved AVF evaluation framework is implemented. Compared with other AVF evaluation tools, AVF computed using the evaluation framework is reduced by 22.6% averagely.

The Comparison

In this section, we compare the three models above from main future, advantage, disadvantage; the details are shown in table 2.

AVF Extension and Research

Currently it has carried out many studies in AVF computing calculations. Based on the current methods, researchers make the computing expend on some new aspects. For example, the method based on indication bit of online [15] and the occupancy-based online AVF computing method [23]. This occupancy-based online AVF computing method can efficiently compute AVF of different structures when the program is operating. The AVFs vary from different types of application and present diversity with time changes [19, 22].

Sridharan et al also propose the concept of Program Vulnerability Factor (PVF) [20, 21]. PVF metric allows insight into the vulnerability of a software resource to hardware faults in a micro-architecture independent way, and can be used to make judgments about the relative reliability of different programs. This method could enable the development of reliability techniques at a compiler or even programming language level. To determine whether PVF is a good predictor of relative vulnerability, Sridharan et al measured the PVF of each program's Architectural Integer Register File (ARF) and compare these values to the AVF of the Integer Physical Register File (PRF). Experiment results show that the PVF and AVF values have a correlation coefficient of 0.98 [20]. In addition, most prior studies have focused on single-threaded micro-architecture, TAN J et al make AVF evaluation methods expand on multi-threaded micro-architecture, and propose the corresponding soft protective measures [24].

Future Trends

In this paper, we have summarized the current research methods, and the future research will be mainly concentrated in the following aspects:

- Most work only considers the case of single-threaded applications currently, AVF show diversity from the system workloads, operating systems, and also changes with the system state. Therefore the appropriate methods and techniques will be the focus of the study about multi-threaded system. There are some researchers conducted a study in this regard [25, 26, 27].

- The current AVF research work is still concentrated on the architecture and micro-architecture, the future research will be expanded to the level of the whole system level and the program level.
- Most of the existing research is about transient faults, but intermittent failures are also the leading causes of system failure. Some research on intermittent failures has been done already [28, 29].
- The current assessment is mainly used in the microprocessor's early design, the future can be used in other areas.

TABLE 2 THE COMPARISON AMONGST THE THREE MODELS

Analysis Model	Main feature	Advantage	Disadvantage
Statistical fault injection [3, 8, 9, 10]	Fault injection into the processor model, analyze the impact on the result of the program	fault injection for any state unit, high precision	Long experiment time, analyze different structures' AVF in the late of design
Litter's Law [11, 12]	Computing the production and propagation of fault based on the numerical analysis	Establish the relationship between the soft error and system MTBF, analyze the contribution of different structures on system reliability	It's complex to analyze and compute all the structures' AVF
Performance mode [2, 13, 14, 16, 17, 18]	Analyzing ACE and Un-ACE	The different structures' AVF values are computed in early stage of design and it's easy to guide the reliability design	It's complex to determine the ACE and Un-ACE

REFERENCES

[1] C. Weaver et al. Techniques to Reduce the Soft Error Rate of a High-Performance Microprocessor. In ISCA, 2004.

[2] Li X, Adve S V, and Bose P, et al. SoftArch: an architecture-level tool for modeling and analyzing soft errors .Proceedings of the International Conference on Dependable Systems and Networks (DSN), Yokohama, Japan, June 2005: 496-505.

[3] N. Wang et al. Characterizing the Effects of Transient Faults on a Modern High-Performance Processor Pipeline. In DSN, 2004.

[4] K. Trivedi. Probability and Statistics with Reliability, Queueing, and Computer Science Applications. Prentice Hall, 1982.

[5] X. Li, S. V. Adve, P. Bose, and J. A. Rivers. Architecture-level soft error analysis: Examining the limits of common assumptions. In DSN '07: Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, pages 266-275, Washington, DC, USA, 2007. IEEE Computer Society.

[6] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin. A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In MICRO 36: Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture, page 29, Washington, DC, USA, 2003. IEEE Computer Society.

[7] Biswas, P. Racunas, R. Cheveresan, J. Emer, S. S. Mukherjee, and R. Rangan. Computing architectural vulnerability factors for address-based structures. In ISCA '05: Proceedings of the 32nd annual international symposium on Computer Architecture, pages 532-543, Washington, DC, USA, 2005. IEEE Computer Society.

[8] Nicholas Wang and Sanjay Patel. Modeling the Effect of Transient Errors on High Performance Microprocessors, Center for Circuits, Systems, and Software (C2S2), 2nd Annual Review, Berkeley, and March 18-19, 2003.

[9] Wu Zhen-ping et al. Sensitivity Analysis for Processor Based on VHDL Fault Injection, Microelectronics & Computer , 2012.

[10] GIL D, GRACIA J, BARAZA J, et al. Analyzing the impact of Intermittent Faults on Microprocessors applying Fault Injection. Design & Test of Computers. IEEE, 2012, 29 (6): 66-73.

[11] E.D.Lazowska, J.Zahorjan, G.S.Graham, and K.C.Sevcik. Quantitative System Performance, Prentice-Hall, Inc., E140nglewood Cliffs, New Jersey, 1984.

[12] LI X, ADVE S V et al. SoftArch: an architecture-level tool for modeling and analyzing soft errors. Dependable Systems and Networks, 2005. DSN 2005. Proceedings. International Conference on. IEEE, 2005: 496-505.

[13] Fu X, Li T, Fortes J a B. Sim-SODA: A Unified Frame work for Architectural Level. Proc of Workshop on Modeling , Benchmarking and Simulation , 2006

[14] Biswas, R. Cheveresan, J. Emer, S. S. Mukherjee, P. B. Racunas, and R. Rangan.: Computing Architectural Vulnerability Factors for Address-Based Structures, In Proceedings of the International Symposium on Computer Architecture, 2005.

[15] X. D. U, S. V. Adve, P. Bose, J. A. Rivers. Online Estimation of Architectual Vulnerability Factor for Soft Errors. In Proceedings of the International Conference of Computer Architecture (ISCA), PP. 341-352, 2008.

[16] Zhou X, Yu J, Li X, et al. Research on reliability evaluation of cache based on instruction behaviour. Journal of Computer Research and Development, 2007, 44(4): 553-559.

[17] Hartl R, Rohatschek A J, Stechele W, et al. Architectural vulnerability factor estimation with backwards analysis. 13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools (DSD), Lille, September 2010: 605-612.

[18] Cheng Yu et al. Research on reliability evaluation of memories for low-cost fault tolerant design. Journal of Electronics & Information Technology , 2011, 11

[19] XU J, TAN Q, LIU W. Estimating the soft error vulnerability of register files via inter-procedural data flow analysis. Theoretical Aspects of Software Engineering (TASE), 2010 4th IEEE International Symposium on .IEEE, 2010

[20] V. Sridharan and D. R. Kaeli. Quantifying software vulnerability. In Workshop on Radiation Effects and Fault Tolerance in Nanometer Technologies (WREFT-1), 2008.

[21] V. Sridharan and D. R. Kaeli. Eliminating Microarchitectural Dependency from Architectural Vulnerability. 2008 IEEE

[22] SALEHI M E, AZAREYVAND et al. Reliability Analysis of Embedded Applications in Non-Uniform Fault Tolerant Processors. Future Information Technology, 2010 5th International Conference on .IEEE, 2010

[23] PAN S, HU Y et al. Online computing and predicting architectural vulnerability factor of microprocessor structures. Dependable Computing, 2009. PRDC'09. 15th IEEE Pacific Rim International Symposium on. IEEE,2009

[24] TAN J, LI Z, FU X. Cost-effective soft error protection for SRAM-based structures in GPGPUS. Proceedings of the ACM International Conference on Computing Frontiers. ACM, 2013:29

[25] DUAN L, PENG L, LI B. Predicting architectural vulnerability on multi-threaded processors under resource contention and sharing. Dependable and Secure Computing, IEEE Transactions on 2013,10

[26] DUAN L, PENG L, LI B. Two-level soft error vulnerability prediction on SMT/CMP architectural. Workload Characterization, 2011 IEEE International Symposium on. IEEE, 2011

[27] TANG L, WANG S, HU J, et al. Characterizing the L1 Date cache's vulnerability to transient errors in chip-multiprocessors. VLSI, 2011 IEEE Computer Society Annual Symposium on. IEEE, 2011

[28] KOTHAWADE S, CHAKRABORTY K, ROY S, et al. Analysis of intermittent timing fault vulnerability. Microelectronics Reliability, 2012, 52(7):1515-1522

[29] RASHID L. Tolerating intermittent hardware errors: characterization, diagnosis and recovery. 2013

Baolong. Guo received his B. S. degree in 1984, the M. S. degree in 1988 and the Ph. D. degree in 1995, all in Communication and Electronic System from Xidian University, Xi'an, and China and is a professor in Xidian University. His research interests are intelligent information processing and video communication.